Finding the Maximal Pose Error in Robotic Mechanical Systems Using Constraint Programming

Nicolas Berger[1], Ricardo Soto[1,2], Alexandre Goldsztejn[1], Stéphane Caro[3], and Philippe Cardou[4]

[1] LINA, CNRS, Université de Nantes, France
[2] Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso, Chile
[nicolas.berger,ricardo.soto,alexandre.goldsztejn]@univ-nantes.fr
[3] IRCCyN, Ecole Centrale de Nantes, France
stephane.caro@irccyn.ec-nantes.fr
[4] Department of Mechanical Engineering, Laval University, Canada
pcardou@gmc.ulaval.ca

Abstract. The position and rotational errors —also called pose errors— of the end-effector of a robotic mechanical system are partly due to its joints clearances, which are the play between their pairing elements. In this paper, we model the prediction of those errors by formulating two continuous constrained optimization problems that turn out to be NP-hard. We show that techniques based on numerical constraint programming can handle globally and rigorously those hard optimization problems. In particular, we present preliminary experiments where our global optimizer is very competitive compared to the best-performing methods presented in the literature, while providing more robust results.

1 Introduction

The accuracy of a robotic mechanical system is a crucial feature for the realization of high-precision operations. However, this accuracy can negatively be affected by position and/or orientation errors, also called pose errors, of the manipulator end-effector. A main source of pose errors is the joints clearances that introduce extra degree-of-freedom displacements between the pairing elements of the manipulator joints. It appears that the lower those displacements, the higher the manufacturing cost and the more difficult the mechanism assembly. Currently, handling this concern is a major research trend in robotics [7, 9, 14]. A way to deal with joints clearances is to predict their impact on the pose errors. To this end, the involved displacements together with a set of system parameters can be modeled as two continuous constrained optimization problems, whose global maximum characterize the maximum pose error. It is therefore mandatory to obtain a rigorous upper bound of this global maximum, which disqualifies the usage of local optimizers. This model can be seen as a sequence of variables

lying in a continuous domain, a set of constraints, and an objective function. It turns out that such a model is computationally hard to solve: Computing time may increase exponentially with respect to the problem size, which in this case depends on the number of manipulator joints.

There exists some research works dealing with joint clearances modeling and error prediction due to joint clearances [7, 9]. However, few of them deal with the solving techniques [14], and no work exists related to the use of constraint programming, a powerful modeling and solving approach. In this paper, we investigate the use of constraint programming for solving such problems. In particular, we combine the classic branch-and-bound algorithm with interval analysis and powerful pruning techniques. The branch-and-bound algorithm allows us to handle the optimization part of the problem, while computing time can be reduced thanks to the pruning techniques. Moreover, the reliability of the process is guaranteed by the use of interval analysis, which is mandatory for the application presented in this paper. The experiments demonstrate the efficiency of the proposed approach w.r.t. state-of-the-art solvers GAMS/BARON [1] and ECL^{i}PS^e [18].

The remaining of this paper is organized as follows. Section 2 presents an example of robot manipulator with the associated model for predicting the impact of the joints clearances on the end-effector pose. A presentation of constraint programming including the implemented approach is given in Sect. 3. The experiments are presented in Sect. 4, followed by the conclusion and future work.

2 Optimization Problem Formulation

In the scope of this paper, we consider serial manipulators composed of n revolute joints, n links and an end-effector. Figure 1 illustrates such a manipulator composed of two joints, named RR-manipulator, as well as a clearance-affected revolute joint. Let us assume that joint clearances appear due to manufacturing errors. Accordingly, the optimization problem aims to find the maximal positional and rotational errors of the manipulator end-effector for a given manipulator configuration.

2.1 End-Effector Pose Without Joint Clearance

In order to describe uniquely the manipulator architecture, i.e. the relative location and orientation of its neighboring joints axes, the Denavit-Hartenberg nomenclature is used [6]. To this end, links are numbered 0, 1, ..., n, the jth joint being defined as that coupling the $(j-1)$st link with the jth link. Hence, the manipulator is assumed to be composed of $n+1$ links and n joints; where 0 is the fixed base, while link n is the end-effector. Next, a coordinate frame $\mathcal{F}_j$ is defined with origin O_j and axes X_j, Y_j, Z_j. This frame is attached to the $(j-1)$st link for $j = 1, \ldots, n+1$. The following screw takes $\mathcal{F}_j$ onto $\mathcal{F}_{j+1}$:

$$\mathbf{S}_j = \begin{bmatrix} \mathbf{R}_j & \mathbf{t}_j \\ \mathbf{0}_3^T & 1 \end{bmatrix}, \tag{1}$$

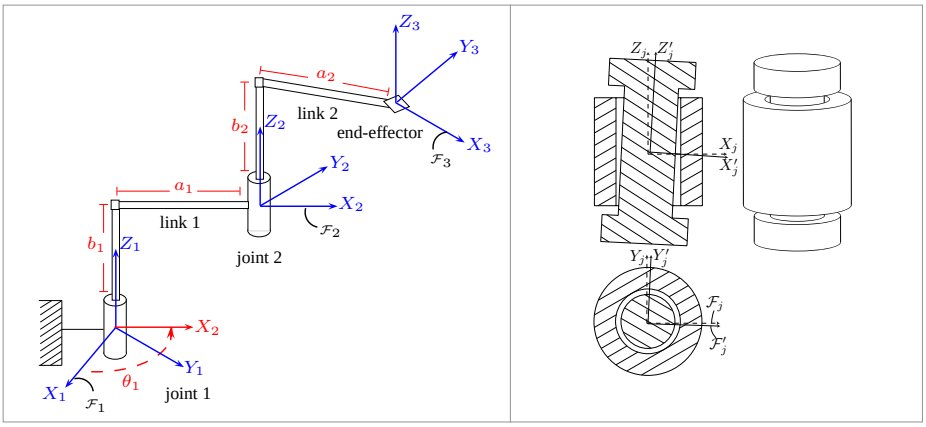

Fig. 1. Left: A serial manipulator composed of two revolute joints. Right: Clearance-affected revolute joint.

where $\mathbf{R}_j$ is a 3×3 rotation matrix; $\mathbf{t}_j \in \mathbb{R}^3$ points from the origin of $\mathcal{F}_j$ to that of $\mathcal{F}_{j+1}$; and $\mathbf{0}_3$ is the three-dimensional zero vector. Moreover, $\mathbf{S}_j$ may be expressed as:

$$\mathbf{S}_j = \begin{bmatrix} \cos\theta_j & -\sin\theta_j \cos\alpha_j & \sin\theta_j \sin\alpha_j & a_j \cos\theta_j \\ \sin\theta_j & \cos\theta_j \cos\alpha_j & -\cos\theta_j \sin\alpha_j & a_j \sin\theta_j \\ 0 & \sin\alpha_j & \cos\alpha_j & b_j \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where α_j, a_j, b_j and θ_j represent respectively the link twist, the link length, the link offset, and the joint angle. Let us notice that a_1, b_1, a_2 and b_2 are depicted in Fig. 1 for the corresponding manipulator whereas α_1 and α_2 are null as the revolute joints axes are parallel.

Provided that the joints are perfectly rigid in all directions but one, that there is no joint clearance, that the links are perfectly rigid and that the geometry of the robotic manipulator is known exactly, the pose of the end-effector with respect to the fixed frame $\mathcal{F}_1$ is expressed as:

$$\mathbf{P} = \prod_{j=1}^{n} \mathbf{S}_j, \tag{3}$$

However, if we consider joint clearances, we must include small errors in Eq. (3).

2.2 Joint-clearance Errors

Taking into account joint clearances, the frame $\mathcal{F}_j$ associated with link $j-1$ is shifted to $\mathcal{F}'_j$. Provided it is small, this error on the pose of joint j with respect to joint $j-1$ may be represented by the small-displacement screw depicted in Eq. (4):

$$\delta\mathbf{s}_j \equiv \begin{bmatrix}\delta\mathbf{r}_j \\ \delta\mathbf{t}_j\end{bmatrix} \in \mathbb{R}^6, \qquad (4) \qquad\qquad \delta\mathbf{S}_j = \begin{bmatrix}\delta\mathbf{R}_j & \delta\mathbf{t}_j \\ \mathbf{0}_3^T & 0\end{bmatrix}, \qquad (5)$$

where $\delta\mathbf{r}_j \in \mathbb{R}^3$ represents the small rotation taking frame $\mathcal{F}_j$ onto $\mathcal{F}'_j$, while $\delta\mathbf{t}_j \in \mathbb{R}^3$ points from the origin of $\mathcal{F}_j$ to that of $\mathcal{F}'_j$. It will be useful to represent $\delta\mathbf{s}_j$ as the 4×4 matrix given with Eq. (5) where $\delta\mathbf{R}_j \equiv \partial(\delta\mathbf{r}_j \times \mathbf{x})/\partial\mathbf{x}$ is the cross-product matrix of $\delta\mathbf{r}_j$. Intuitively, clearances in a joint are best modeled by bounding its associated errors below and above. Assuming that the lower and upper bounds are the same, this generally yields six parameters that bound the error screw $\delta\mathbf{s}_j$. Accordingly, the error bounds are written as:

$$\delta r_{j,X}^2 + \delta r_{j,Y}^2 \leq \delta\beta_{j,XY}^2, \quad (6) \qquad\qquad \delta t_{j,X}^2 + \delta t_{j,Y}^2 \leq \delta b_{j,XY}^2, \quad (8)$$

$$\delta r_{j,Z}^2 \leq \delta\beta_{j,Z}^2, \quad (7) \qquad\qquad \delta t_{j,Z}^2 \leq \delta b_{j,Z}^2, \quad (9)$$

where $\delta\mathbf{r}_j \equiv [\delta r_{j,X}\ \delta r_{j,Y}\ \delta r_{j,Z}]^T$ and $\delta\mathbf{t}_j \equiv [\delta t_{j,X}\ \delta t_{j,Y}\ \delta t_{j,Z}]^T$.

2.3 End-Effector Pose With Joint Clearances

Because of joints clearances, the end-effector frame $\mathcal{F}_{n+1}$ is shifted to $\mathcal{F}'_{n+1}$. From [16], the displacement taking frame $\mathcal{F}_j$ onto $\mathcal{F}'_j$ is given by the matrix exponential of $\delta\mathbf{S}_j$, $e^{\delta\mathbf{S}_j}$. As a result, the screw that represents the pose of the shifted end-effector may be computed through the kinematic chain as:

$$\mathbf{P}' = \prod_{j=1}^{n} e^{\delta\mathbf{S}_j}\mathbf{S}_j, \qquad (10)$$

where screw $\mathbf{P}'$ takes frame $\mathcal{F}_1$ onto $\mathcal{F}'_{n+1}$ when taking errors into account.

2.4 The End-Effector Pose Error Modeling

In order to measure the error on the pose of the moving platform, we compute the screw $\Delta\mathbf{P}$ that takes its nominal pose $\mathcal{F}_{n+1}$ onto its shifted pose $\mathcal{F}'_{n+1}$ through the kinematic chain, namely:

$$\Delta\mathbf{P} = \mathbf{P}^{-1}\mathbf{P}' = \prod_{j=n}^{1}\mathbf{S}_j^{-1}\prod_{j=1}^{n}\left(e^{\delta\mathbf{S}_j}\mathbf{S}_j\right). \qquad (11)$$

From [4], it turns out that $\Delta\mathbf{P}$ may as well be represented as a small-displacement screw $\delta\mathbf{p}$ in a vector form, namely,

$$\delta\mathbf{p} = \sum_{j=1}^{n} \left(\prod_{k=n}^{j} \mathrm{adj}(\mathbf{S}_k)^{-1} \right) \delta\mathbf{s}_j, \quad \text{with} \quad \mathrm{adj}(\mathbf{S}_j) \equiv \begin{bmatrix} \mathbf{R}_j & \mathbf{O}_{3\times3} \\ \mathbf{T}_j\mathbf{R}_j & \mathbf{R}_j \end{bmatrix}$$

being the adjoint map of screw $\mathbf{S}_j$ and $\mathbf{T}_j \equiv \partial(\mathbf{t}_j \times \mathbf{x})/\partial\mathbf{x}$ the cross-product matrix of $\mathbf{t}_j$.

2.5 The Maximum End-Effector Pose Error

Let $\delta\mathbf{p}$ be expressed as $[\delta\mathbf{p}_r\ \delta\mathbf{p}_t]^T$ with $\delta\mathbf{p}_r \equiv [\delta p_{r,X}\ \delta p_{r,Y}\ \delta p_{r,Z}]^T$ and $\delta\mathbf{p}_t \equiv [\delta p_{t,X}\ \delta p_{t,Y}\ \delta p_{t,Z}]^T$ characterizing the rotational and translational errors of the manipulator end-effector, respectively. In order to find the maximal pose errors of the end-effector for a given manipulator configuration, we need to solve two optimization problems:

$$\begin{aligned} \textbf{max} \quad & \|\delta\mathbf{p}_r\|_2, \qquad (12) \\ \textbf{s.t.} \quad & \delta r_{j,X}^2 + \delta r_{j,Y}^2 - \delta\beta_{j,XY}^2 \leq 0, \\ & \delta r_{j,Z}^2 - \delta\beta_{j,Z}^2 \leq 0, \\ & j = 1, \ldots, n \end{aligned} \qquad \begin{aligned} \textbf{max} \quad & \|\delta\mathbf{p}_t\|_2, \qquad (13) \\ \textbf{s.t.} \quad & \delta r_{j,X}^2 + \delta r_{j,Y}^2 - \delta\beta_{j,XY}^2 \leq 0, \\ & \delta r_{j,Z}^2 - \delta\beta_{j,Z}^2 \leq 0, \\ & \delta t_{j,X}^2 + \delta t_{j,Y}^2 - \delta b_{j,XY}^2 \leq 0, \\ & \delta t_{j,Z}^2 - \delta b_{j,Z}^2 \leq 0, \\ & j = 1, \ldots, n \end{aligned}$$

where $\|.\|_2$ denotes the 2-norm. The maximum rotational error due to joint clearances is obtained by solving problem (12). Likewise, the maximum point-displacement due to joint clearances is obtained by solving problem (13). It is noteworthy that the constraints of the foregoing problems are defined with Eqs. (6)–(9). Moreover, it appears that those problems are nonconvex quadratically constrained quadratic (QCQPs). Although their feasible sets are convex—all the constraints of both problems are convex—their objectives are convex, making the computation of their global maximum NP-Hard.

3 Constraint Programming

3.1 Definitions

Constraint Programming (CP) is a programming paradigm that allows one to solve problems by formulating them as a Constraint Satisfaction Problem (CSP). We are mainly interested in Numerical CSP (NCSP) whose variables belong to continuous domains. This formulation consists of a sequence of variables lying in a domain and a set of constraints that restrict the values that the variables can take. The goal is to find a variable-value assignment that satisfies the whole set of constraints. Formally, a NCSP $\mathcal{P}$ is defined by a triple $\mathcal{P} = \langle \mathcal{X}, [\mathbf{x}], \mathcal{C} \rangle$ where:

- $\mathcal{X}$ is a vector of variables $(x_1, x_2, \ldots, x_n)$.
- $[\mathbf{x}]$ is a vector of real intervals $([x_1], [x_2], \ldots, [x_n])$ such that $[x_i]$ is the domain of x_i.
- $\mathcal{C}$ is a set of constraints $\{c_1, c_2, \ldots, c_m\}$. In the scope of this paper, we focus on inequality constraints, i.e., $c(\mathbf{x}) \iff g(\mathbf{x}) \leq 0$ where $g : \mathbb{R}^n \to \mathbb{R}$ is assumed to be a differentiable function.

A solution of a CSP is a real vector $\mathbf{x} \in [\mathbf{x}]$ that satisfies each constraint, i.e. $\forall c \in \mathcal{C}, c(\mathbf{x})$. The set of solutions of the CSP $\mathcal{P}$ is denoted by $\text{sol}(\mathcal{P})$. This definition can be extended to support optimization problems by also considering a cost function f. Then we want to find the element of $\text{sol}(\mathcal{P})$ that minimizes or maximizes the cost function.

3.2 Interval Analysis

The modern interval analysis was born in the 60's with [15] (see [17, 10] and references therein). Since, it has been widely developed and is today one central tool in the rigorous resolution of NCSPs (see [3] and extensive references).

Intervals, interval vectors and interval matrices are denoted using brackets. Their sets are denoted respectively by $\mathbb{IR}$, $\mathbb{IR}^n$ and $\mathbb{IR}^{n \times m}$. The elementary functions are extended to intervals in the following way: Let $\circ \in \{+, -, \times, /\}$ then $[x] \circ [y] = \{x \circ y : x \in [x], y \in [y]\}$ (division is defined only for denominators that do not contain zero). E.g. $[a, b] + [c, d] = [a + c, b + d]$. Also, continuous functions $f(x)$ with one variable are extended to intervals using the same definition: $f([x]) = \{f(x) : x \in [x]\}$, which turns to be an interval as f is continuous. When numbers are represented with a finite precision, the previous operations cannot be computed in general. The outer rounding is then used so as to keep valid the interpretations. For example, $[1, 2] + [2, 3]$ would be equal to $[2.999, 5.001]$ if rounded with a three decimal accuracy.

Then, an expression which contains intervals can be evaluated using this interval arithmetic. The main property of interval analysis is that such an interval evaluation gives rise to a superset of the image through the function of the interval arguments. For example, the interval evaluation of expression $x(y - x)$ is $[x] \times ([y] - [x])$ and contains $\{x(y - x) : x \in [x], y \in [y]\}$. In some cases (e.g. when the expression contains only one occurrence of each variable), this enclosure is optimal.

Given an n-ary constraint c and a box $[\mathbf{x}] \in \mathbb{R}^n$, a contractor for c will contract the box $[\mathbf{x}]$ without losing any solution of c. Some widely used contractors are based on the 2B-consistency (also called hull-consistency) or the box consistency [12, 2], which are pruning methods similar to the well known arc-consistency [13] in the context of discrete CSPs. They are both applied to one constraint at a time, hence suffering of the usual drawbacks of the locality of their application. We encapsulate this notion in the function $\text{Contract}_{\mathcal{C}}([\mathbf{x}])$ which uses the constraints $\mathcal{C}$ to prune the box $[\mathbf{x}]$. Thus, the result is a new box $[\mathbf{x}'] \subseteq [\mathbf{x}]$ that satisfies $\mathbf{x} \in [\mathbf{x}] \wedge (\forall c \in \mathcal{C}, c(\mathbf{x})) \Rightarrow \mathbf{x} \in [\mathbf{x}']$, i.e. no solution was lost during the pruning. This property, rigorously achieved thanks to the correct

rounding of interval arithmetic, allows the CP framework to provide rigorous proofs of mathematical statements.

3.3 The Branch and Bound Algorithm

NCSPs are usually solved using a branch and prune algorithm. A basic branch and prune algorithm is described by Algorithm 1. Its input is a set of constraints and an initial box domain. It interleaves pruning (Line 4) and branching (Line 5) to output a set of boxes that sharply covers the solution set: Due to the property satisfied by the function $\text{Contract}_{\mathcal{C}}([\mathbf{x}])$, Algorithm 1 obviously maintain the property $\mathbf{x} \in [\mathbf{x}] \wedge (\forall c \in \mathcal{C}, c(\mathbf{x})) \Rightarrow \mathbf{x} \in \cup\mathcal{L}$. The stopping criterion is usually the size of the boxes in $\mathcal{L}$, the algorithm stopping when every box got smaller than a fixed precision. We have used the branch and prune algorithm implemented in RealPaver [8] which basically proceeds as Algorithm 1 does.

A branch and prune algorithm can be modified to a branch and bound algorithm that handles minimization problems (maximization problems are handled similarly). Such a simple branch and bound algorithm is described by Algorithm 2. The cost function is an additional input, and maintains an upper bound on the global minimum in the variable m (which is initialized to $+\infty$ at the beginning of the search at Line 2). This upper bound is used to discard parts of the search space whose cost is larger than it by adding the constraint $f(\mathbf{x}) \leq m$ to the set of constraints (Line 5). Finally, the upper bound is updated at Line 6 by searching for feasible points inside the current box $[\mathbf{x}]$ and when such points are found, by evaluating the cost function at these points. In our current implementation, some random points are generated inside $[\mathbf{x}]$ and the constraints are rigorously checked using interval arithmetic before updating the upper bound[5]. The branch and bound algorithm maintains an enclosure of the global minimum, that converges to the global minimum provided that feasible points are found during the search (which is guaranteed in the case of inequality constraints that are not singular).

An efficient implementation of the branch and bound algorithm requires the list of boxes $\mathcal{L}$ to be carefully handled: It has to be maintained sorted w.r.t. a lower bound of the objective evaluated for each box. So, each time a box $[\mathbf{x}]$ is inserted into $\mathcal{L}$, the interval evaluation of the objective $f([\mathbf{x}])$ is computed and the lower bound of this interval evaluation is used to maintain the list sorted. Then, Line 4 extracts the first box of $\mathcal{L}$ so that most promising regions of the search space are explored first, leading to drastic improvements. Another advantage of maintaining $\mathcal{L}$ sorted is that its first box contains the lowest objective value over all boxes. Therefore, we use this value and check its distance to the current upper bound m and stop the algorithm when the absolute precision of the global minimum has reached a prescribed value.

[5] More elaborated branch and prune algorithms usually use a local search to find good feasible points, but preliminary experiments presented in Section 4 and performed using this simple random generation of potential feasible points already showed good performances.

Algorithm 1
Input: $\mathcal{C} = \{c_1, \ldots, c_m\}$, $[\mathbf{x}]$
1 $\mathcal{L} \leftarrow \{[\mathbf{x}]\}$
2 **While** $\mathcal{L} \neq \emptyset$ **and** $\neg$stop_criteria **do**
3 $([\mathbf{x}], \mathcal{L}) \leftarrow$**Extract**$(\mathcal{L})$
4 $[\mathbf{x}] \leftarrow$**Contract**$_{\mathcal{C}}([\mathbf{x}])$
5 $\{[\mathbf{x}'], [\mathbf{x}'']\} \leftarrow$**Split**$([\mathbf{x}])$
6 $\mathcal{L} = \mathcal{L} \cup \{[\mathbf{x}'], [\mathbf{x}'']\}$
7 **End While**
8 **Return**$(\mathcal{L})$

Algorithm 2
Input: f, $\mathcal{C} = \{c_1, \ldots, c_m\}$, $[\mathbf{x}]$
1 $\mathcal{L} \leftarrow \{[\mathbf{x}]\}$
2 $m \leftarrow +\infty$
3 **While** $\mathcal{L} \neq \emptyset$ **and** $\neg$stop_criteria **do**
4 $([\mathbf{x}], \mathcal{L}) \leftarrow$**Extract**$(\mathcal{L})$
5 $[\mathbf{x}] \leftarrow$**Contract**$_{\mathcal{C} \cup \{f(\mathbf{x}) \leq m\}}([\mathbf{x}])$
6 $m \leftarrow$**Update**$([\mathbf{x}], f)$
7 $\{[\mathbf{x}'], [\mathbf{x}'']\} \leftarrow$**Split**$([\mathbf{x}])$
8 $\mathcal{L} = \mathcal{L} \cup \{[\mathbf{x}'], [\mathbf{x}'']\}$
9 **End While**
10 **Return**$(\mathcal{L}, m)$

Fig. 2. Left: The branch and prune algorithm. Right: The branch and bound algorithm

The branch and bound algorithm described above has been implemented on top of RealPaver [8]. This implementation is used for the experiments presented in the next section.

4 Experiments

We have performed a set of experiments to analyze the performance of our approach solving the pose error problem. We compare it with GAMS/BARON [1] and ECL^{i}PS^e [18]. GAMS/BARON is a widely used system for mathematical programming and optimization[6] and ECL^{i}PS^e is one of the few CP systems supporting continuous optimization problems.

We have tested 8 models, 4 translations (T) and 4 rotation (R) models. For both types of model we consider from 2 to 5 joints. Table 1 depicts the results obtained. Columns 1 and 2 show the number of joints and the type of problem. Columns 3 to 5 depict relevant data about the problem size (number of variables, number of constraints, and number of arithmetic operations in the objective function). Columns 5 to 10 depict the solving times using GAMS, RealPaver (including 2 different filtering techniques), and ECL^{i}PS^e. Experiments were run on a 3 Ghz Pentium D with 2 GB of RAM running Ubuntu 9.04. All solving times are the best of five runs.

The results show that our approach is faster in almost all cases. In smaller problems such as 2R, 2T, 3R and 3T, RealPaver exhibits great performance. It can even be 100 times faster than GAMS. Such a faster convergence can be explained on one hand by the efficient work done by the filtering techniques HC4 and BC4 (based on hull and box consistencies respectively) and on the other hand by the fact that there is no need for an accurate computation for this particular problem. In fact, we do not have to go beyond a precision of 10^{-2}, making the interval computations less costly than usual. Moreover, the

[6] In our version, LP and NLP solving are respectively done by CPLEX and MINOS.

Table 1. Solving times (seconds)

#joints	Type	Problem Size			GAMS	RealPaver		ECL^{i}PS^e
		#var	#ctr	#op		HC4	BC4	
2	T	12	8	28	0.08	0.004	0.004	>60
2	R	6	4	18	0.124	0.004	0.004	>60
3	T	18	12	135	0.124	0.008	0.016	t.o.
3	R	9	6	90	0.952	0.004	0.008	t.o.
4	T	24	16	374	0.144	0.152	0.168	t.o.
4	R	12	8	205	2.584	0.02	0.02	t.o.
5	T	30	20	1073	0.708	>60	>60	t.o.
5	R	15	10	480	9.241	0.26	0.436	t.o.

impact on the solving time of the used consistency is not surprising: using HC4 is faster than BC4, as expected when dealing with constraints having variables not occurring multiple times [5].

In bigger problems such as 4R and 5R, RealPaver remains faster. However, when the problem size arises, in particular when the number of variables exceeds 20, the convergence starts to be slow. This is a common phenomenon we can explain thanks both to the exponential in problem size complexity of this kind of branch-and-bound solving algorithms and to the fact that the objective function itself grows exponentially with the number of variables it involves. Nevertheless, we believe that solving times are reasonable with regard to the complexity and size of the problems as well as the techniques used. But solving times may actually not be the point to emphasize here: on the contrary, it may rather be important to discuss the reliability of the solutions given by GAMS/BARON. It was already noted in [11] that BARON is not reliable in general. We have also verified using RealPaver that the optimum enclosure computed by BARON is unfeasible.

Finally, it is worth noticing that although RealPaver as well as ECL^{i}PS^e use CP techniques, ECL^{i}PS^e is completely outperformed even for the smallest of our instances. It is not surprising, given that RealPaver was designed to solve problems of this kind whereas ECL^{i}PS^e is a more generic and extensible tool.

5 Conclusion and Future Work

In this paper we have shown how to efficiently solve the pose error problem by using constraint programming. We have modeled the problem as a NCSP, and then shown how to turn a CP classical branch-and-prune resolution scheme into a branch-and-bound algorithm, dedicated to the solving of continuous constrained optimization problems. Finally, we presented experimental results for several instances of the problem, and showed that our approach was very competitive with respect to some state-of-the-art CP and optimization tools.

It is important to notice that this approach is not specific to robotics, it is indeed applicable to any problem where rigorous computation and numerical reliability are required. Moreover, in the future we should be able to design new pruning criteria in order to accelerate the convergence of the optimization process. This way, we should tackle the scalability issue and thus be able to solve faster the large instances of the problem.

References

1. GAMS. http://www.gams.com/ (Visited 10/2009).
2. F. Benhamou, D. Mc Allester, and P. Van Hentenryck. CLP(Intervals) Revisited. In *Proceedings of ILPS*, pages 124–138. MIT Press Cambridge, MA, USA, 1994.
3. F. Benhamou and W.J. Older. Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *Journal of Logic Programming*, 32(1):1–24, 1997.
4. P. Cardou and S. Caro. The Kinematic Sensitivity of Robotic Manipulators to Manufacturing Errors. Technical report, IRCCyN, Internal Report No RI2009_4, 2009.
5. H. Collavizza, F. Delobel, and M. Rueher. Comparing Partial Consistencies. *Reliable Computing*, 5(3):213–228, 1999.
6. J. Denavit and R.S. Hartenberg. A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. *ASME J. Appl. Mech*, 23:215–221, 1955.
7. A. Fogarasy and M. Smith. The Influence of Manufacturing Tolerances on the Kinematic Performance of Mechanisms. In *Proc. Inst. Mech. Eng., Part C: J. Mech. Eng. Sci.*, pages 35–47, 1998.
8. L. Granvilliers and F. Benhamou. Algorithm 852: RealPaver: an Interval Solver Using Constraint Satisfaction Techniques. *ACM Trans. Math. Softw.*, 32(1):138–156, 2006.
9. C. Innocenti. Kinematic Clearance Sensitivity Analysis of Spatial Structures With Revolute. *ASME J. Mech. Des.*, 124:52–57, 2002.
10. L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis with Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer-Verlag, 2001.
11. Yahia Lebbah, Claude Michel, and Michel Rueher. An Efficient and Safe Framework for Solving Optimization Problems. *Journal of Computational and Applied Mathematics*, 199(2):372–377, 2007. Special Issue on Scientific Computing, Computer Arithmetic, and Validated Numerics (SCAN 2004).
12. O. Lhomme. Consistency Techniques for Numeric CSPs. In *Proceedings of IJCAI*, pages 232–238, 1993.
13. A. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99–118, 1977.
14. J. Meng, D. Zhang, and Z. Li. Accuracy Analysis of Parallel Manipulators With Joint Clearance. *ASME J. Mech. Des.*, 131, 2009.
15. R. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs N. J., 1966.
16. R. Murray, Z. Li, and S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC, Boca Raton, Fl., 1994.
17. A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge Univ. Press, 1990.
18. M. Wallace, S. Novello, and J. Schimpf. ECLiPSe: A Platform for Constraint Logic Programming. Technical report, IC-Parc, Imperial College, London, 1997.